МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ІНФОРМАЦІЙНА СИСТЕМА ПІДБОРУ СПИСКУ ВІДТВОРЕННЯ МУЗИКИ У ТОРГОВО-РОЗВАЖАЛЬНОМУ ЦЕНТРІ

Здобувач освіти гр. ІН – 82 **Сєріков Руслан**

Завідувач кафедри,
доктор технічних наук,
професор **Довбиш Анатолій**

Науковий керівник,
Кандидат техн.наук **Берест Олег**

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2022 р.

ЗАВДАННЯ
до випускної роботи

здобувача вищої освіти четвертого курсу, групи ІН-82 спеціальності «122 – Комп'ютерні науки» денної форми навчання Сєрікова Руслана Валерійовича.

Тема: «Інформаційна система підбору списку відтворення музики у торгово-розважальному центрі»

Затверджена наказом по СумДУ

№_____ від _____2022 р.

Зміст пояснювальної записки: 1) літературний огляд за обраною тематикою роботи; 2) постановка завдання для розробки; 3) вибір оптимальних інструментів для розробки інформаційної системи; 4) практична реалізація.

Дата видачі завдання "_____"_____2022 р.

Керівник випускної роботи _____ Олег БЕРЕСТ

Завдання прийняв до виконання _____ Руслан СЄРІКОВ

РЕФЕРАТ

Записка: 43 стор., 23 рис., 1 додаток, 8 джерел.

Об’єкт дослідження — інформаційна система підбору списку відтворення музики у торгово-розважальному центрі

Мета роботи — дослідити та розробити інформаційне та програмне забезпечення підбору списку відтворення музики у торгово-розважальному центрі

Методи дослідження — технології створення веб-застосунків; технології підбору музики;

Результати — розроблено інформаційну систему для підбору списку відтворення музики у торгово-розважальному центрі у вигляді веб-застосунку. Створений застосунок простий у використанні для будь-якої цільової аудиторії, виконує поставлені задачі та має можливість для масштабування.

PYTHON, DJANGO, WEB-PROGRAMMING, SPOTIFY

ЗМІСТ

ВСТУП

Наше життя неможливо уявити без музики. Завдяки ній, ми надихаємося, радіємо, сумуємо, мотивуємося – вона відкриває широкий спектр емоцій в нас.

Окрім цього, вона є не лише розвагою, а також корисним інструментом в багатьох сферах життя – особливо, в маркетингу. Музика є важливою складовою, адже правильно підібрані композиції можуть залучити більше нових або постійних клієнтів до об’єкту – кафе, спортивна зала, торгово-розважальний центр. В той же час, неправильно підібрана музика навпаки може відштовхнути від об’єкту споживачів, що в свою чергу вплине на продажі, які вплинуть на подальший прибуток та розвиток.

Як це реалізувати? Звичайно, можна витратити певні кошти на залучення маркетологів та аудіовізуальних спеціалістів, які вручну підберуть підходящі для об’єкта плейлисти на будь-яку подію: свято, дощ, сніг тощо. Проте, варто розуміти, що тренди в музичній сфері частіше за все швидкоплинні, тому плейлист доведеться час від часу оновлювати, а це нові витрати на спеціалістів, які можуть кожного разу збільшуватися.

Саме тому, щоб оптимізувати дані процеси, було розроблено автоматичні системи з підбору плейлистів для об’єктів. Їх перевага в тому, що оплата відбувається лише за користування ними, можливо також за техніку для програвання музики (сервер, аудіотехніка). На відміну від замовлення

профільних спеціалістів, витрати набагато менші через автоматизовані процеси.

Метою нашої дипломної роботи буде розробити аналогічну інформаційну систему, яка формуватиме список для програвання музики у торгово-розважальному центрі, спираючись на зовнішні фактори: який сьогодні день, яка погода на вулиці тощо.

1 АНАЛІЗ ВІДОМИХ РІШЕНЬ

Для створення власної інформаційної системи, ми маємо розглянути аналогічні рішення. Відомими компаніями в Україні, що надають такі рішення, є Bolero [1], Retail Music [2] та Esthetic Sound [3]. Проте, так як ці компанії мають на меті надати послуги та заробити за них кошти, ми не можемо повною мірою проаналізувати функціонал їх продуктів, адже їм невигідно надавати такі рішення в open-source простір, а для того щоб отримати demo-версію, потрібно пройти повноцінну перевірку і лише після цього ми змогли би проаналізувати схожі рішення.

Тому ми розглянемо відомі стрімінгові сервіси, які мають функцію підбору плейлистів, а головне – ліцензійну музику.

1.1 Огляд існуючих рішень

1.1.1 Spotify

Першим ми розглянемо стрімінговий сервіс Spotify. Даний продукт починався як шведський стартап у 2008 році [4], а відтепер працює у багатьох країнах світу (окрім росії та Куби).

Головною перевагою сервісу є те, що він працює за моделлю freemium, тобто користувач може легально та безкоштовно слухати музику від провідних лейблів, але має певні обмеження, за зняття яких доведеться доплатити.

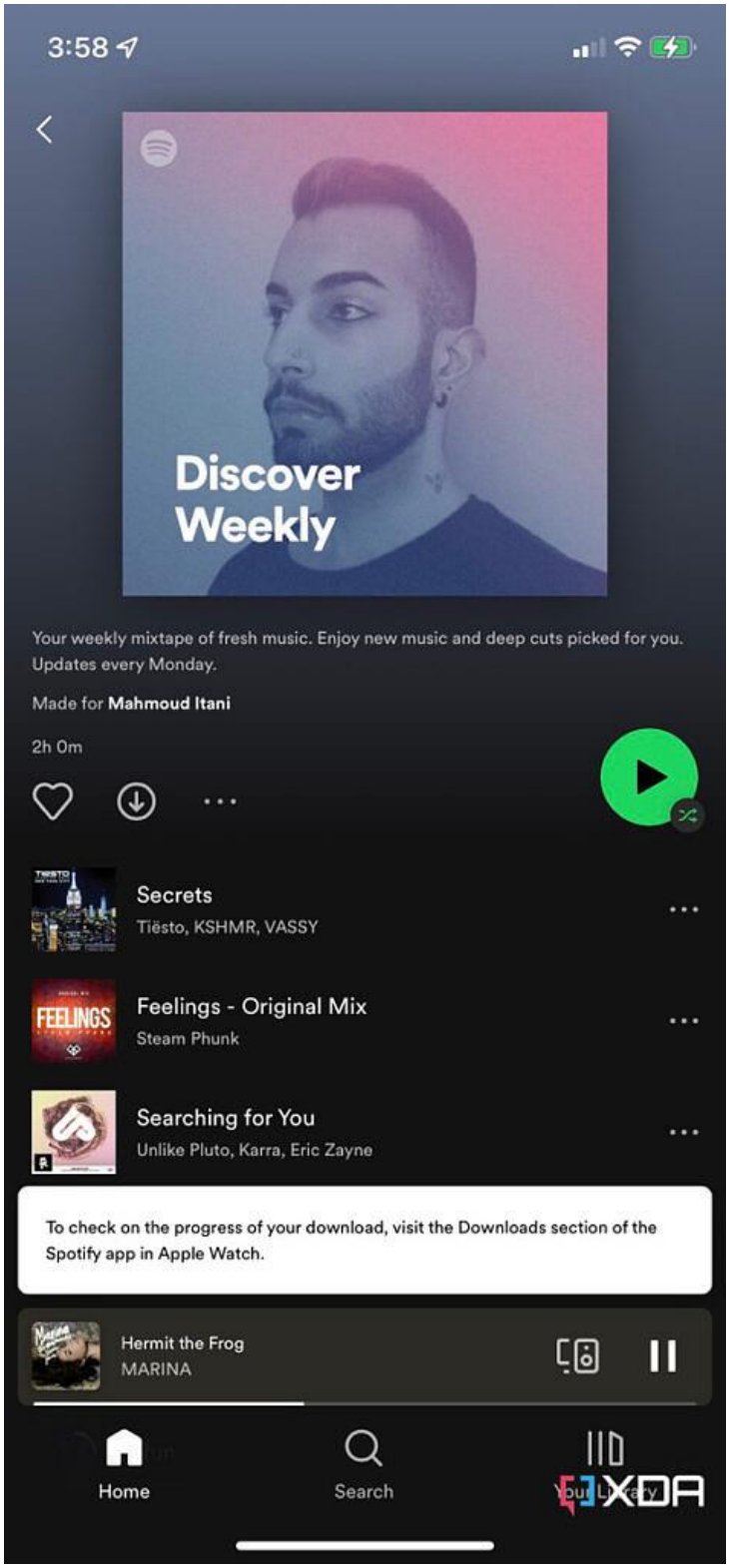

Рисунок 1.1 - Застосунок Spotify на iOS пристрої

Також перевагою сервісу є його крос-платформеність, тобто він доступний для використання майже на будь-якому пристрої: Windows, macOS, Linux, Android та iOS.

При безпосередньому тестуванні сервісу, було виявлено що Spotify має можливість побудови різного роду плейлистів для прослуховування. І дані списки програвання дійсно є різноманітними – від жанрових до «Плейлист для сонячної погоди», «Плейлист для тренування» тощо. Такі плейлисти будуються згідно отриманих даних про користувачів: їх вподобання, коли саме слухають музику, яка погода коли вони слухають ту чи іншу композицію тощо. Власне кажучи, Spotify не лише чудовий стрімінговий сервіс, а також розвинений тренувальний полігон для data scientist'ів, адже має власний API – опис способів, за допомогою яких застосунок розробника може взаємодіяти зі Spotify.

1.1.2 Youtube Music

Наступним сервісом для розгляду буде Youtube Music. Його було розроблено у листопаді 2015 як сервіс потокового аудіо відеохостингу Youtube.

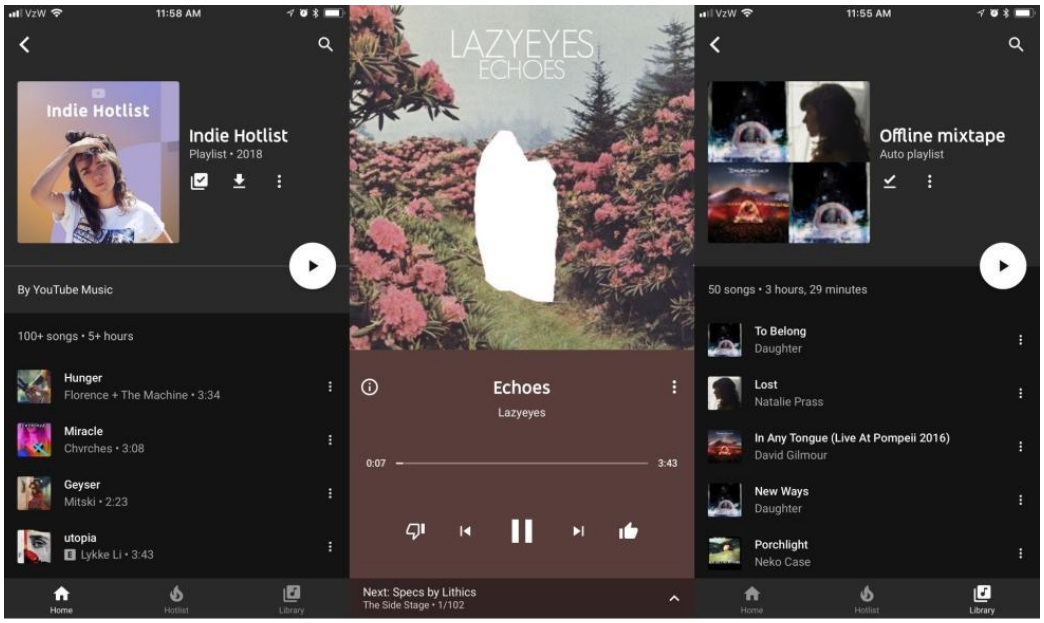

Рисунок 1.2 - Зовнішній вигляд музичного сервісу Youtube Music

Звичайно, даний сервіс не має настільки багатого функціоналу, як Spotify, але має ключову перевагу – даний сервіс працює як умовне розширення сервісу Youtube, а тому плейлисти на основі ваших вподобань формуються не лише під час безпосередньої взаємодії з сервісом Youtube Music, а навіть коли ви переглядаєте та слухаєте музичні відео на основному відеохостингу. Тобто, прослухавши низку композицій на відеохостингу під час роботи за комп'ютером, коли ви виходите на вулицю, надягаєте навушники та заходите до Youtube Music – ви вже маєте сформовані плейлисти для тренування, яскравої прогулянки тощо.

Спільним у Youtube Music та Spotify є те, що обидва стримінгові сервіси працюють за моделлю freemium, проте у випадку з Youtube Music ви отримуєте більше додаткових функцій, а саме:

1. Відключення реклами у відеохостингу Youtube та сервісі Youtube Music;
2. Можливість фонового прослуховування або перегляду відео-, музичного контенту.

Таким чином, не дивлячись на те що надається лише 2 пункти додаткових послуг, ви отримуєте як переваги на відеохостингу, так і на музичному сервісі, на відміну від Spotify.

Для того, щоб побудувати мінімально життєздатну версію нашої інформаційної системи, ми не будуватимемо власний стрімінговий сервіс «з нуля», а використаємо один з вже розглянутих нами продуктів.

Для досягнення потрібної мети було обрано сервіс Spotify, адже він має зрозумілий API, безліч параметрів які стосуються плейлиста або безпосередньо композиції, тому завдяки таким факторам нам легше буде розробити інформаційну систему підбору списку програвання музики у торгово-розважальному центрі.

1.2 Постановка задачі

Після проведеного аналізу, було визначено мету нашої дипломної роботи – проектування та розробка інформаційної системи підбору списку програвання музики у торгово-розважальному центрі. Для досягнення поставленої мети було визначено такий перелік задач:

1) Ознайомлення з Spotify API;
2) Проектування алгоритму підбору списку відтворення музики та його інтеграція у веб-застосунок;
3) Розробка веб-застосунку;
4) Програмування алгоритму підбору списку відтворення музики та його інтеграція у веб-застосунок;
5) Тестування сторонніми користувачами;

2 ВИБІР ОСНОВНИХ КОМПОНЕНТІВ ЗАСТОСУНКУ ТА ЇХ РЕАЛІЗАЦІЯ

2.1 Проектування алгоритму підбору списку відтворення музики

Напередодні розробки нашого продукту, потрібно виконати його проектування. Ми поділимо дане завдання на декілька частин, а саме: проектування алгоритму підбору списку відтворення музики, веб-застосунку з інтеграцією алгоритму підбору, а також взаємодія користувач-застосунок.

Таке проектування «по атомах» дозволить нам розглянути всі процеси, що потрібні для досягнення наших цілей, а потім інтерпретувати їх у вигляді коду.

Почнемо з проектування алгоритму підбору списку відтворення музики. Перш за все, потрібно розглянути, за якими саме критеріями можливий підбір музики. На рисунку 2.1 представлено діаграму чинників, за допомогою яких можливо підібрати композиції для програвання.

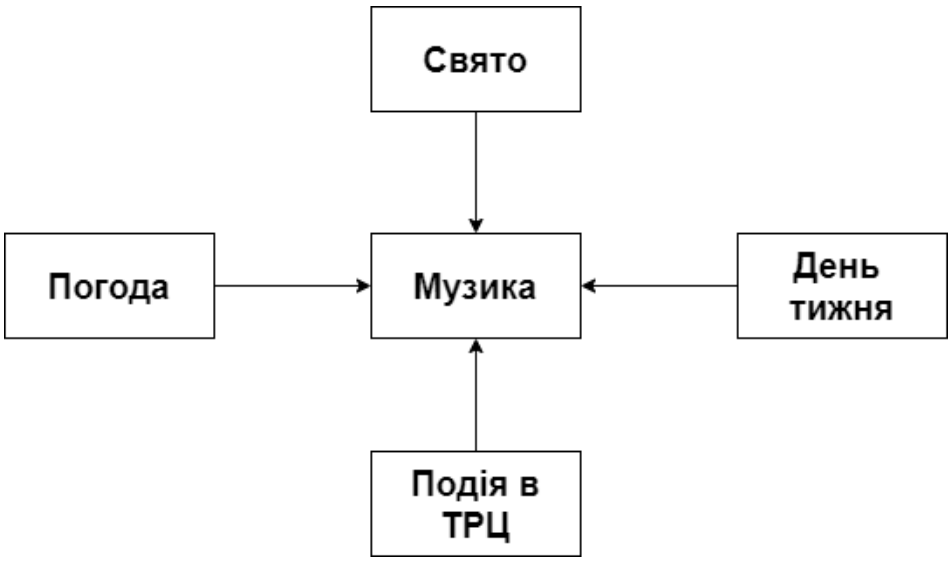

Рисунок 2.1 - Діаграма чинників підбору музичних композицій

Тобто, ми маємо справу з 4 основними чинниками: погода, свято в країні/місті, день тижня та подією в торгово-розважальному центрі. Варто зазначити, що ці фактори пов’язані між собою, тобто не можна у дощову погоду програвати щось на літню тематику, незважаючи на те що зараз субота в червні.

Даний факт потрібно врахувати при побудові алгоритму. Окрім цього, потрібно також пам’ятати, що погода може змінюватися кожної години, разом з цим можемо спостерігати помилки у прогнозах, тому на перших порах не вдасться на початку дня, згідно зібраних даних, сформувати список відтворення на найближчі 24 години. Це доведеться робити кожної години, але з часом ми можемо зберігати так звані шаблони подій («Дощова погода», «День незалежності», «Новий рік», «Чорна п’ятниця» тощо) і замість постійної перебудови списків відтворення, ми будуватимемо лише послідовність таких списків згідно подій на більш тривалі проміжки часу.

На рис.2.2 представлено початковий алгоритм підбору списку відтворення.

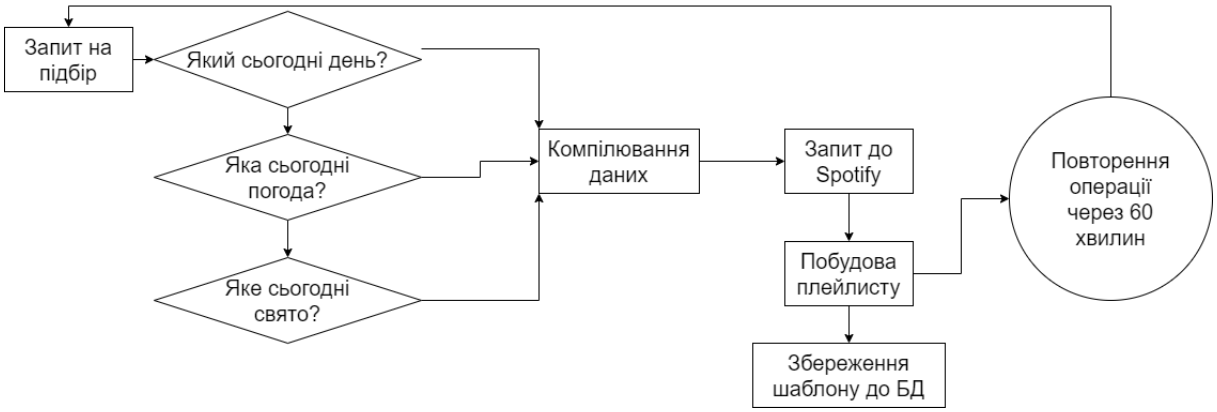

Рисунок 2.2 - Початковий алгоритм підбору списку відтворення

Проаналізуємо детальніше даний алгоритм. Сервіс робить запит на підбір списку програвання композицій. Сервіс проходить та збирає дані по трьох питаннях: який сьогодні день, яка сьогодні погода, яке сьогодні свято.

Всі ці дані формуються в один запит і він передається на Spotify, який згідно наданих параметрів вибудовує плейлист. Сформований список відтворення зберігається до бази даних з такими параметрами: які чинники були під час побудови списку відтворення, як саме виглядає список відтворення згідно чинників. Потім, через 60 хвилин, дана операція повторюється, проте оновлюється інформація лише за одним питанням – яка сьогодні погода, адже день та наявність свята нам вже відомі.

Такий алгоритм є складним для сервера і справа не в кількості даних, а в тому як часто він буде виконуватися. Якщо кожні 60 хвилин збирати дані, то згодом це може привести до сповільнення серверу, що в свою чергу приведе до його непрацездатності.

Саме тому в алгоритмі є пункт «Збереження шаблону до БД» і саме тому алгоритм є початковим. Свята кожного року повторюються, погода також буває сонячною, дощовою, сніжною, похмурою тощо. Певний час сервіс буде навчатися і потім можна перейти до оптимізованого алгоритму, який представлено на рис.2.3.

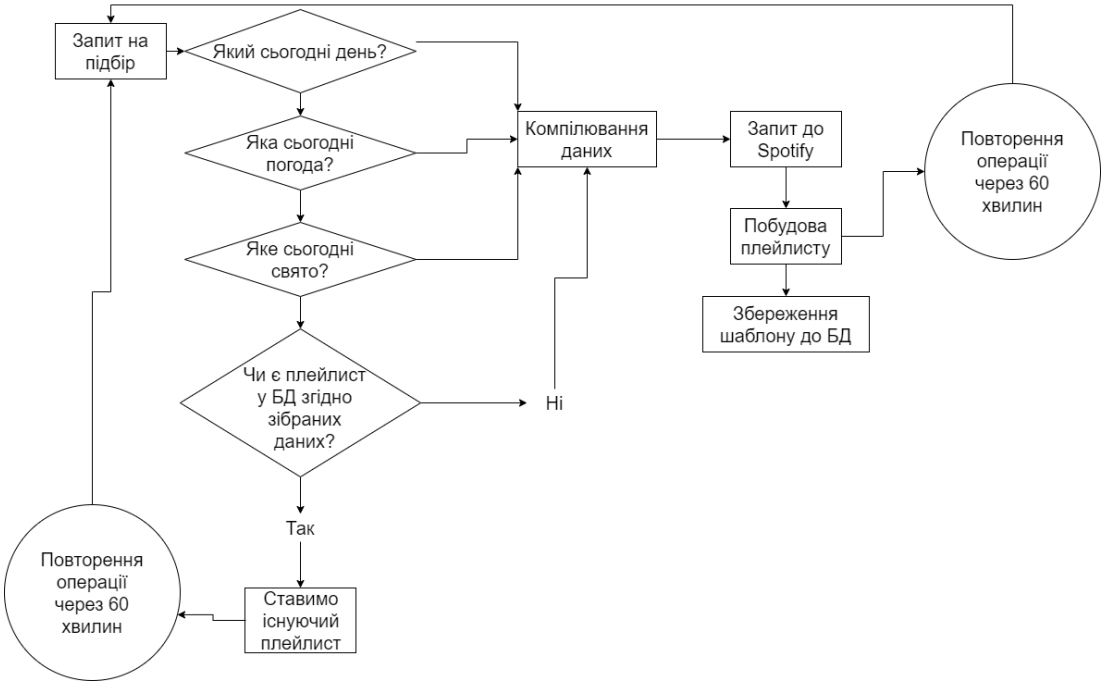

Рисунок 2.3 - Оптимізований алгоритм підбору списку відтворення

Розглянемо алгоритм більш детально. Починається він так само, як і перший алгоритм: запит на підбір, збирання відповідей на питання, але тепер є роздоріжжя. Згідно зібраних відповідей, відбувається перевірка, чи є вже схожий шаблон у базі даних сервісу. Якщо так – сервіс ставить на програвання вже існуючий плейлист та повторює операцію через 60 хвилин. Якщо ні – виконуються ті самі дії, що й у початковому алгоритмі. Не дивлячись на повторення дій через 60 хвилин, ми вже можемо зробити висновок що такий алгоритм дещо розвантажить наш сервер. Завдяки вже існуючим шаблонам плейлистів, алгоритм не формуватиме запит на Spotify, не очікуватиме відповіді від нього та не конвертуватиме отриману відповідь у повноцінний плейлист. Навпаки, все відбувається локально: формується запит з даними, відбувається пошук у локальній базі даних і ставиться на програвання вже існуючий список відтворення композицій.

Згодом, коли вже збирається достатня кількість шаблонів, ми ще більше можемо оптимізувати алгоритм підбору списку відтворення музики. На рис.2.4 представлено повторно оптимізований алгоритм створення плейлисту.

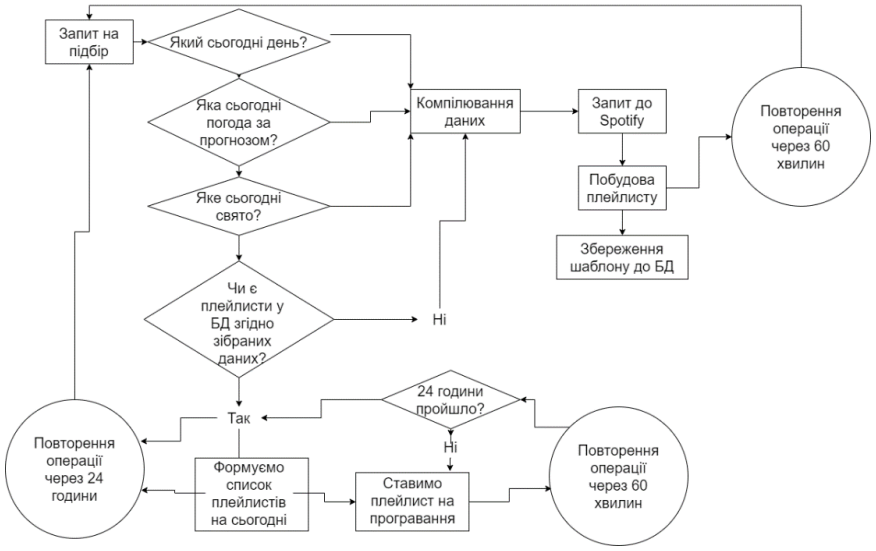

Рисунок 2.4 - Повторно оптимізований алгоритм списку відтворення музики

Дивлячись на діаграму, спершу може здаватися що як таких змін в алгоритмі немає. Проте, ми бачимо що тепер ми питаємо не про погоду на найближчу годину, а її прогноз. Згідно зібраних відповідей, сервіс шукає плейлисти за кожним із шаблонів і якщо їх знаходить, то вибудовує список плейлистів не на 60 хвилин, а на 24 години. Кожної години ставиться новий список відтворення з побудованого списку та перевіряється умова, чи пройшло 24 години. Якщо ні – ставиться новий плейлист, якщо так – повторюється операція формування списків відтворення музики, але вже на наступні 24 години. Таким чином, операція формування плейлистів відтепер відбувається один раз на добу і відбувається перевірка лише на те, чи пройшло 24 години та поставлення наступного, вже існуючого списку відтворення композицій. Завдяки цьому алгоритму, сервіс має ще менше навантаження, що позитивно впливає на його працездатність та витривалість.

Таким чином, ми побудували алгоритм підбору списку відтворення музики і тепер можемо перейти до проектування безпосередньо веб-застосунку, який матиме у собі даний алгоритм.

2.2 Проектування веб-застосунку з використанням побудованого алгоритму

Спроектувавши «серце» нашого продукту, ми можемо перейти до проектування веб-застосунку. Майбутній продукт повинен бути простим та зручним у використанні – тобто таким, щоб для його налаштування та експлуатації не доведеться залучати висококваліфікованого розробника. Ми поділимо проектування веб-застосунку на такі етапи:

1. Структура основних сторінок
2. Функції кожної зі сторінок
3. Алгоритм взаємодії застосунку із сервером
4. Проектування бази даних

Так як ми маємо на меті побудувати продукт, що буде зрозумілий широкій цільовій аудиторії, то наповнення сторінками не має бути перевантаженим. На рис.2.5 представлено діаграму зі сторінками нашого веб-застосунку.

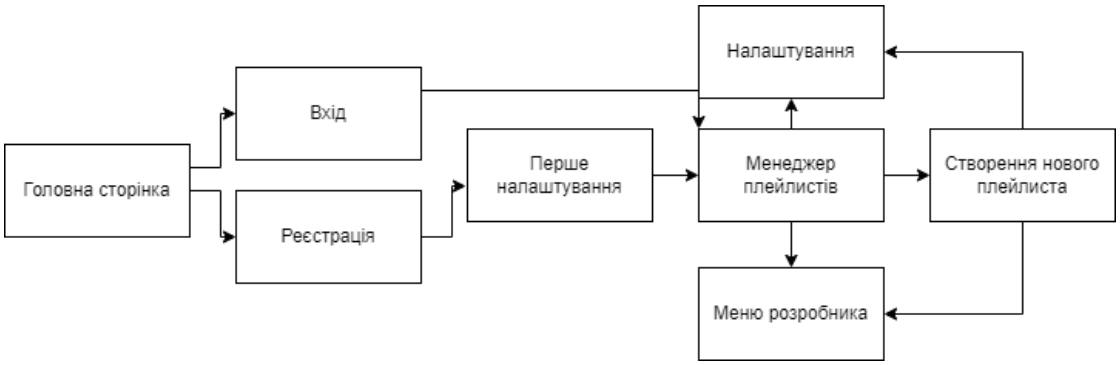

Рисунок 2.5 - Діаграма сторінок веб-застосунку

Зустрічатиме користувача головна сторінка, на якій він матиме два варіанти: увійти на вже існуючий акаунт або зареєструватися. Після реєстрації, користувача зустрічатиме сторінка першого налаштування. На рис.2.6 представлено деталізацію сторінки «Перше налаштування».

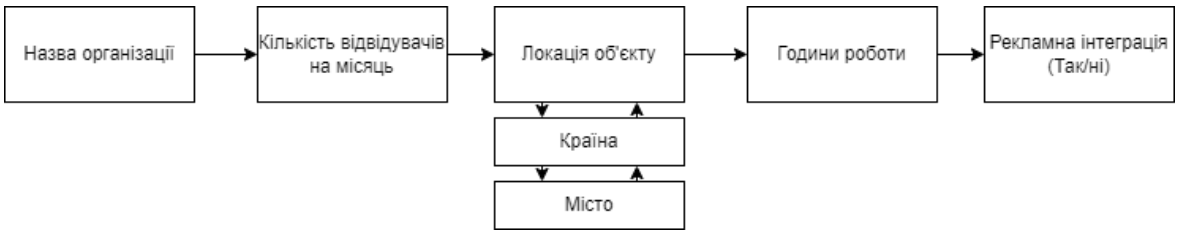

Рисунок 2.6 - Деталізація сторінки «Перше налаштування»

Під час першого налаштування запитуються основні дані про об’єкт, а саме:

- Назва організації;
- Кількість відвідувачів на місяць;
- Локація об’єкту (країна, місто);
- Години роботи;
- Рекламна інтеграція.

Розглянемо окремі пункти, які запитуються під час першого налаштування. Локація об’єкту потрібна для того, щоб:

А) до об’єкта прив’язати список свят, які відбуваються в країні або місті;

Б) при побудові плейлисту, сервіс знав звідки брати інформацію про погоду.

Також є питання про години роботи і це також є корисним для сервісу, адже завдяки цьому він зможе вибудувати правильний графік побудови плейлистів. Тобто, зробити це не під час роботи ТРЦ, що може спричинити технічні несправності в системі та залишити об’єкт без музики, а зробити це у неробочі години, що допоможе вчасно виявити недоліки, якщо такі будуть.

Останній пункт стосується рекламної інтеграції. В ньому ви можете обрати, чи будете інтегрувати власні рекламні треки під час програвання плейлистів чи ні. Дана інформація буде враховуватись під час побудови списків програвання композицій та у випадку, якщо потрібна рекламна інтеграція, буде інтегрувати вашу рекламу між композиціями.

Тепер перейдемо до сторінки «Менеджер плейлистів». На рис.2.7 представлено деталізацію даної сторінки.

Рисунок 2.7 - Деталізація сторінки «Менеджер плейлистів»

Дане меню є простим та не перевантаженим, що позитивно впливає на розширення цільової аудиторії продукту. В ньому користувач може переглянути список наявних плейлистів, які вже було зібрано; список шаблонів для підбору плейлистів (погода, свято, день тижня); перейти до створення нового плейлиста.

Виникає питання: навіщо для інформаційної системи, яка автоматично створює потрібні плейлисти, потрібна функція ручного створення? Насправді, в ній може бути потреба у таких випадках:

А) сервер, на який ми робимо запит для підбору композицій (в нашому випадку це Spotify), не відповідав протягом певного часу;

Б) замість того, щоб чекати автоматичного створення списків відтворення за різноманітними шаблонами, користувач може самотужки налаштовувати такі шаблони, давати запити на підбори плейлистів і в майбутньому це може полегшити роботу сервісу;

В) тестування сервісу, якщо щось йде не так.

Тобто, дана функція, згідно розглянутих випадків, є корисною і тому має бути в нашому сервісі.

Якщо казати про налаштування, то в цьому випадку ми не будемо прибігати до деталізації сторінки, адже вона призначена для зміни параметрів, які ми надали під час реєстрації або першого налаштування. Можливо, у ТРЦ змінилося місцерозташування; години роботи тепер інші; контактні дані змінилися тощо.

Сторінка «Створення плейлиста» також не потребує деталізації, адже підбір списку відтворення відбувається автоматично згідно вже побудованого алгоритму.

Тепер ми перейдемо до проектування взаємодії застосунку із сервером. На рис.2.8 зображено діаграму взаємодії між сервісом та сервером.

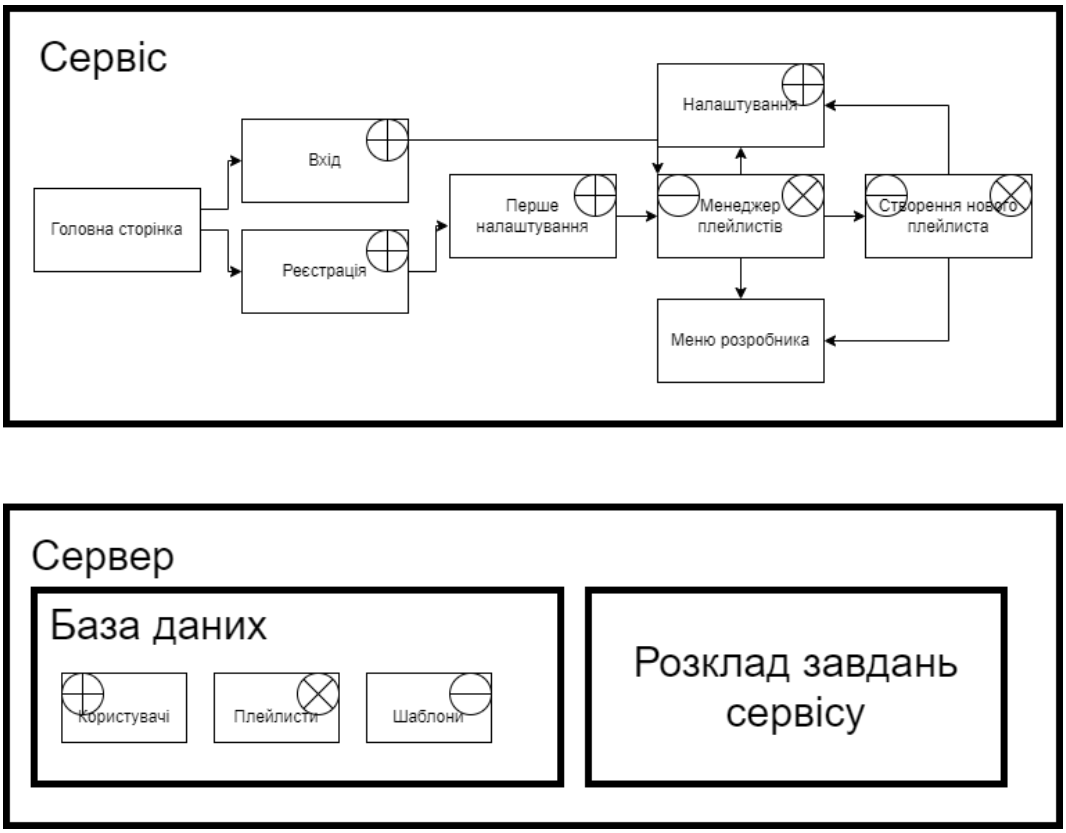

Рисунок 2.8 - Діаграма наповнення та взаємодії сервісу та серверу

Для спрощеного представлення, було обрано 3 умовні позначки, які показують що та як взаємодіє між сервісом та сервером. Наприклад, сторінка «Вхід» взаємодіє з таблицями в базі даних групи «Користувачі», а «Менеджер плейлистів» взаємодіє з таблицями в базі даних групи «Плейлисти». Також сервер, окрім бази даних, матиме ще одну базу даних, яка відповідатиме за розклад завдань сервісу. Тобто, сервіс в певні інтервали буде звертатися до цієї БД та переглядати, які завдання потрібно виконати.

Тепер ми можемо перейти до етапу проектування бази даних веб-застосунку.

2.3 Проектування бази даних веб-застосунку

Тепер перейдемо до проектування бази даних нашого веб-застосунку. Так як веб-застосунок матиме велику кількість різноманітних даних, то ми поділимо наші діаграми БД на декілька фрагментів. На рис.2.9 представлено ER-діаграму таблиць user, organization, country, holiday.

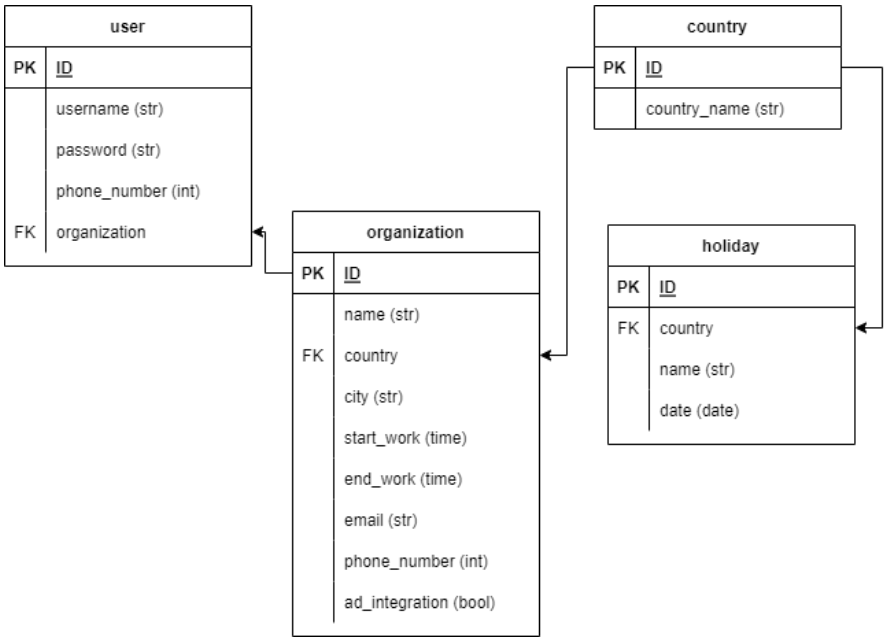

Рисунок 2.9 - ER-діаграма таблиць user, organization, country, holidays

Розглянемо кожну з таблиць окремо. Почнемо з таблиці user. Вона відповідає за дані користувачів та має в собі такі рядки:

1. ID (PK)
2. Username (str) – ім’я користувача
3. Password (str) – пароль користувача
4. Phone_number (int) – мобільний телефон користувача.
5. Organization (FK) – організація, до якої належить користувач.

Останній рядок є зовнішнім ключем, адже у однієї організації може бути декілька користувачів. Тепер перейдемо до таблиці organization, яка містить в собі такі рядки:

1. ID (PK)
2. Name (str) – назва організації
3. Country (FK) – країна, в якій розташовано торгово-розважальний центр.
4. City (str) – місто, в якому розташовано торгово-розважальний центр.
5. Start_work (time) – година, о котрій об’єкт розпочинає свою роботу.
6. End_work (time) – година, о котрій об’єкт завершує свою роботу.
7. Email (str) – адреса електронної скриньки організації
8. Phone_number (int) – номер телефону організації.

9. Ad_integration (bool) – рядок, в якому вказується, чи потрібна рекламна інтеграція у списках відтворення музики.

Країна тут також у вигляді зовнішнього ключа, адже ми маємо окрему таблицю країн та свят, які в них відбуваються. Розглянемо таблицю holiday, де знаходяться свята в країнах. Вона містить в собі такі рядки:

1. ID (PK)
2. Country (FK) – країна, в якій свято.
3. Name (str) – назва свята.
4. Date (date) – дата, коли буде свято.

Ця таблиця буде використовуватися під час формування запиту до Spotify для отримання потрібного плейлиста на певний період часу.

Тепер перейдемо до проектування таблиць для плейлистів. На рис.2.10 представлено ER-діаграму таблиць playlist_name, playlist, playlist_pattern.

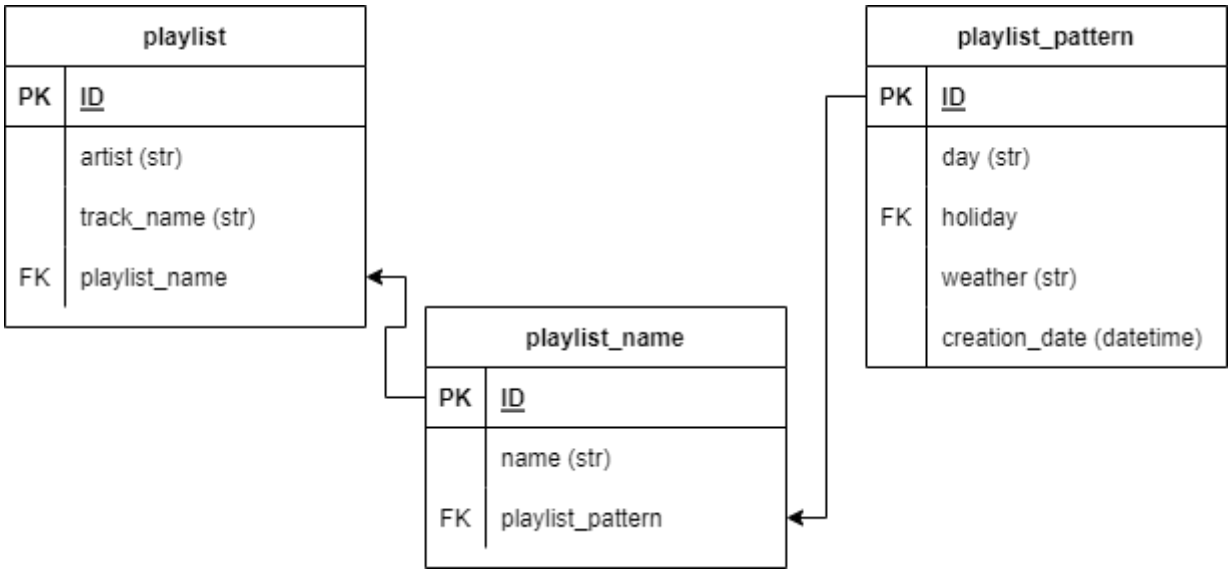

Рисунок 2.10 - ER-діаграма таблиць playlist, playlist_name, playlist_pattern

Розглянемо в цьому випадку кожну з таблиць окремо. Почнемо з playlist, яка має такі рядки:

1. ID (PK)
2. Artist (str) – виконавець композиції
3. Track_name (str) – назва композиції

4. Playlist_name (FK) – список відтворення, до якого належить композиція.

Тепер перейдемо до таблиці playlist_name, яка містить в собі такі рядки:

1. ID (PK)
2. Name (str) – назва плейлиста
3. Playlist_pattern (FK) – шаблон, до якого належить список відтворення.

І останньою таблицею є playlist_pattern. Вона містить в собі шаблони, за якими створюються плейлисти, а ці шаблони містять в собі такі дані:

1. ID (PK)
2. Day (str) – день, в який створено плейлист
3. Holiday (FK) – свято, в яке створений плейлист (Null, якщо свята в цей день не було).
4. Weather (str) – дані про погоду, в яку був створений плейлист.
5. Creation_date (datetime) – дата, коли було створено шаблон.

Отже, ми виконали повноцінне проектування нашого майбутнього застосунку і тепер можемо перейти безпосередньо до його розробки.

3 КОМП'ЮТЕРНА РЕАЛІЗАЦІЯ ПРОЕКТУ ТА ТЕСТУВАННЯ

3.1 Вибір середовища розробки

Для розробки даної ІС було обрано мову програмування Python, фреймворк для створення веб-застосунків Django та СУБД SQLite. Розглянемо кожну з цих складових окремо.

Мова програмування Python є високорівневою, інтерпретованою та містить в собі втілені принципи ООП, проте розробник також може будувати застосунки з використанням функціонального програмування. Дана мова програмування широко використовується у багатьох сферах життя, але якщо розглядати саме web-програмування, то її відомими користувачами є Youtube, Instagram, Spotify та Reddit. Дані сервіси кажуть самі за себе, якщо у нас постає питання про швидкодію та ефективність Python.

Для розробки нашої ІС, з-поміж багатьох фреймворків для веб-програмування мовою Python, було обрано Django.

Django дозволяє розроблювати сайти за допомогою так званих «модулів», тобто кожний пакунок функцій = модуль. Це неабияка перевага, адже завдяки модульності, Django надає можливість не лише швидко розробити мінімально життєздатний продукт, а також не менш швидко та ефективно його масштабувати у разі потреби. Також даний фреймворк має численну кількість користувачів, а отже для нього розроблено безліч плагінів та додатків, які спрощують розробку веб-застосунків.

В якості СУБД було обрано SQLite, яка є компактною та має в собі реалізації стандарту SQL-92.

Дану СУБД було обрано для розробки мінімально життєздатної ІС, адже завдяки її компактності ми не витрачатимемо час на додаткове налаштування СУБД, як це відбувається з іншими відомими аналогами, але поставлені задачі все одно будуть виконані. До того ж, фреймворк Django за замовчуванням

працює з SQLite, а також має ORM – особливість, яка дозволяє взаємодіяти з базою даних, не відходячи від синтаксису мови Python.

У майбутньому, SQLite буде замінено на PostgreSQL, а також буде додано Redis та Celery для планування задач ІС. Розглянемо майбутні складові нашої інформаційної системи окремо.

PostgreSQL є також реляційною СУБД, але має ряд переваг над SQLite. По-перше, широкий вибір структур та типів даних, яким не може похизуватися будь-яка інша СУБД. PostgreSQL підтримує uuid, грошовий, перераховуємий, геометричний, бінарний тип даних, мережеві адреси, бітові рядки, масиви, композитні типи та діапазони тощо. По-друге, дана СУБД має мінімальні обмеження за розмірами таблиць. Наприклад, у PostgreSQL одна таблиця може займати до 32 ТБ даних, рядок до 1.6 ТБ, а поле – до 1 ГБ. Ну і найголовніша перевага – це те, що PostgreSQL є не просто реляційною, а об’єктно-реляційною СУБД, що робить її гнучкою, на відміну від конкурентів.

Також, в майбутньому для планування задач в нашій ІС, ми використаємо Celery та Redis.

Celery є асинхронною чергою задач, що дозволяє запланувати декілька подій одночасно та не допустити того, щоб одна подія вплинула на іншу через можливі внутрішні конфлікти. Завдяки асинхронному підходу, Celery не навантажує сервер та ефективно розподіляє ресурси.

Для зберігання планування ми використаємо Redis, який дозволить зберігати заплановані задачі не в оперативній пам’яті, а в постійній. Це є оптимальним рішенням, адже після відновлення роботи або перезавантаження серверу, оперативна пам’ять повністю очищується, проте завдяки Redis після однієї з двох таких операцій, сервер повернеться до виконання запланованих задач без зайвих проблем.

3.2 Програмна реалізація застосунку

Для початку розробки, повинен бути встановлений Python версії, не старіше ніж 3.8 та віртуальне середовище, яке ми повинні активувати. Якщо дані кроки було виконано, отже варто зробити додаткові налаштування:

1. Встановити фреймворк Django за допомогою команди pip install Django, а також бібліотеки pyowm та spotipy за допомогою аналогічної команди.
2. Створюємо проект за допомогою команди django-admin startproject musicmaster.
3. Створюємо основний модуль нашого проекту за допомогою команди python manage.py startapp musicmaster.

Після виконання цих кроків, ми повинні не забути додати наш модуль до файлу settings.py у розділ INSTALLED_APPS. Це потрібно для того, щоб наш застосунок та всі його модулі запустилися коректно, адже Django перевіряє цей розділ та запускає кожен модуль, який в ньому вказаний. Тепер наш розділ INSTALLED_APPS виглядає так:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'musicmaster'
]
```

Тепер переходимо до моделювання бази даних. Якщо в минулому розділі ми виконували її проектування, то тепер у Django ми створюємо моделі таблиць нашої бази даних. Це одна з переваг даного фреймворку, що ми можемо створити власну базу даних без використання SQL-синтаксису. Достатньо ознайомитися з документацією, визначитися які типи даних нам

потрібні, запрограмувати нашу майбутню базу даних, а Django все зробить самостійно. Моделі нашої бази даних виглядають так:

```
from django.db import models


class Country(models.Model):
    country_name = models.CharField(max_length=25)


class Holidays(models.Model):
    country = models.ForeignKey(Country,
on_delete=models.CASCADE)
    name = models.CharField(max_length=25)
    date = models.DateField()


class Organization(models.Model):
    name = models.CharField(max_length=25)
    country = models.ForeignKey(Country,
on_delete=models.CASCADE)
    city = models.CharField(max_length=25)
    start_work = models.TimeField()
    end_work = models.TimeField()
    email = models.EmailField()
    phone_number = models.IntegerField()
    ad_integration = models.BooleanField()


class User(models.Model):
    username = models.CharField(max_length=30)
    password = models.CharField(max_length=30)
    phone_number = models.IntegerField()
    organization = models.ForeignKey(Organization,
on_delete=models.CASCADE)


class PlaylistPattern(models.Model):
    day = models.DateField()
    holiday = models.ForeignKey(Holidays,
on_delete=models.CASCADE)
    weather = models.CharField(max_length=30)
    creation_date = models.DateTimeField()
```

```
class PlaylistName(models.Model):
    name = models.CharField(max_length=30)
    pattern = models.ForeignKey(PlaylistPattern,
on_delete=models.CASCADE)


class Playlist(models.Model):
    artist = models.CharField(max_length=30)
    track_name = models.CharField(max_length=30)
    playlist_name = models.ForeignKey(PlaylistName,
on_delete=models.CASCADE)
```

Після моделювання нашої бази даних, переходимо до її безпосереднього створення у нашому проекті. Тут Django також має приємну особливість, адже після моделювання, достатньо всього лише написати дві команди для створення таблиць у базі даних:

1. Python manage.py makemigrations – збирає дані про моделі таблиць для бази даних, інтерпретує чистий SQL-скрипт у мову програмування Python, готує файли для подальшого створення таблиць.
2. Python manage.py migrate – шукає файли, що створені під час команди makemigrations, запускає їх, робить запити на створення таблиць у БД, знову інтерпретуючи мову Python в чистий SQL-скрипт, слідкує за перебігом створення таблиць.

Ще однією ключовою особливістю Django є те, що вже «з коробки» він має заготовлені шаблони для потрібних у сервісі панелей. Для цього достатньо лише знати, як саме з цими шаблонами працювати. Наприклад, для відображення даних з БД нам достатньо лише створити файл admin.py та запрограмувати в ньому представлення наших таблиць у БД. Код після таких дій виглядатиме таким чином:

```
from django.contrib import admin

from .models import Holidays, Country, User,
Organization, PlaylistPattern, PlaylistName, Playlist
```

```
@admin.register(Country)
class CountryAdmin(admin.ModelAdmin):
    list_display = "country_name"


@admin.register(Holidays)
class CountryAdmin(admin.ModelAdmin):
    list_display = ("country",
                    "name",
                    "date")


@admin.register(Organization)
class OrganizationAdmin(admin.ModelAdmin):
    list_display = ("name",
                    "country",
                    "city",
                    "start_work",
                    "end_work",
                    "email",
                    "phone_number",
                    "ad_integration")


@admin.register(User)
class UserAdmin(admin.ModelAdmin):
    list_display = ("username",
                    "password",
                    "phone_number",
                    "organization")


@admin.register(PlaylistPattern)
class PlaylistPatternAdmin(admin.ModelAdmin):
    list_display = ("day",
                    "holiday",
                    "weather",
                    "creation_date")


@admin.register(PlaylistName)
class PlaylistNameAdmin(admin.ModelAdmin):
```

```
    list_display = ("name",
                    "pattern")


@admin.register(Playlist)
class PlaylistAdmin(admin.ModelAdmin):
    list_display = ("artist",
                    "track_name",
                    "playlist_name")
```

Розберемо по порядку, що саме міститься в цьому коді.

```
from django.contrib import admin

from .models import Holidays, Country, User,
Organization, PlaylistPattern, PlaylistName, Playlist
```

В цьому шматку ми імпортуємо моделі бази даних та модуль для адміністраторської панелі, який вже у Django з коробки.

```
  @admin.register(Organization)
```

Додаємо до кожного з класів декоратор – структурний паттерн, що дає можливість об’єктам динамічно розширювати функціональність.

```
  class OrganizationAdmin(admin.ModelAdmin):
    list_display = ("name",
                    "country",
                    "city",
                    "start_work",
                    "end_work",
                    "email",
                    "phone_number",
                    "ad_integration")
```

Проводимо ініціалізацію класу, який наслідується від admin.ModelAdmin і в ньому вказуємо, які саме рядки з таблиці потрібно показати в адміністраторській панелі.

І так робимо з кожною таблицею, яка знаходиться у нас в СУБД. Таким чином, тепер ми можемо проводити операції з даними у БД згідно CRUD (Create, Read, Update, Delete).

Також для того, щоб ми могли потрапити до адміністраторської панелі нашого сервісу, ми повинні створити користувача. Для цього достатньо виконати команду python manage.py createsuperuser, виконати кроки які вимагаються і все, користувача створено.

Таким чином, ми створили базову інформаційну систему підбору списку відтворення музичних композицій і тепер ми можемо перейти до тестування сервісу.

3.3 Тестування розробленого застосунку

Тепер ми можемо перейти до етапу тестування нашого застосунку. Для цього ми запускаємо наш сервіс за допомогою python manage.py runserver.

Якщо помилок у коді не було допущено – отже, нас зустріне таке повідомлення:

```
Performing system checks...

Watching for file changes with StatReloader
System check identified no issues (0 silenced).
June 12, 2022 - 18:25:52
Django version 4.0.5, using settings 'musicmaster.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

З цього повідомлення ми можемо зрозуміти, що наш застосунок пройшов усі внутрішні перевірки, використовує версію Django 4.0.5 та потрапити до нашого сервісу можливо за адресою http://127.0.0.1:8000/. Переходячи за посиланням, на рисунку 3.1 ми можемо побачити, що нас зустрічає вітальне повідомлення від фреймворку про те, що застосунок запущено та він працює успішно.

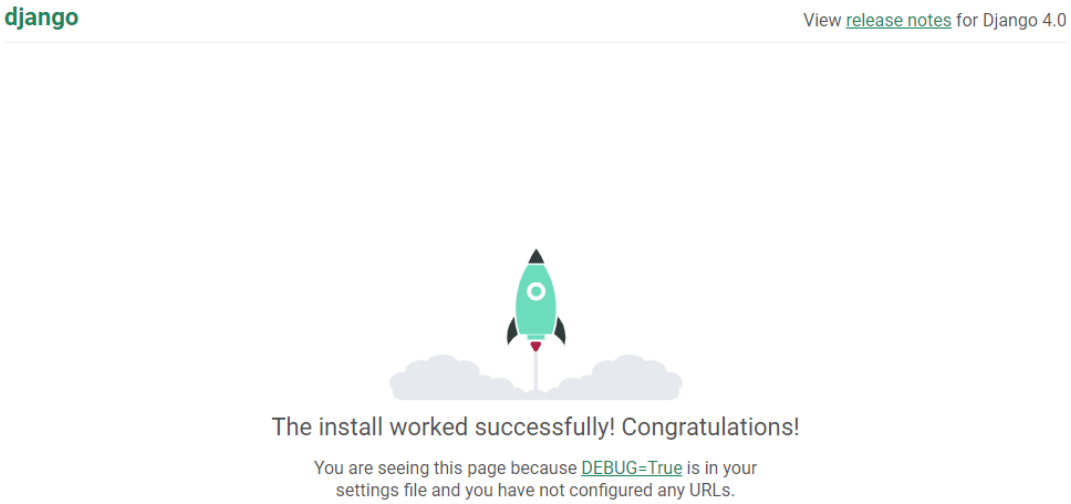

Рисунок 3.1 - Вітальне повідомлення фреймворку Django

Для потрапляння до нашого сервісу, потрібно додати до посилання ключове слово /admin. Після його додавання та натискання кнопки «Enter», ми опиняємося на сторінці авторизації.

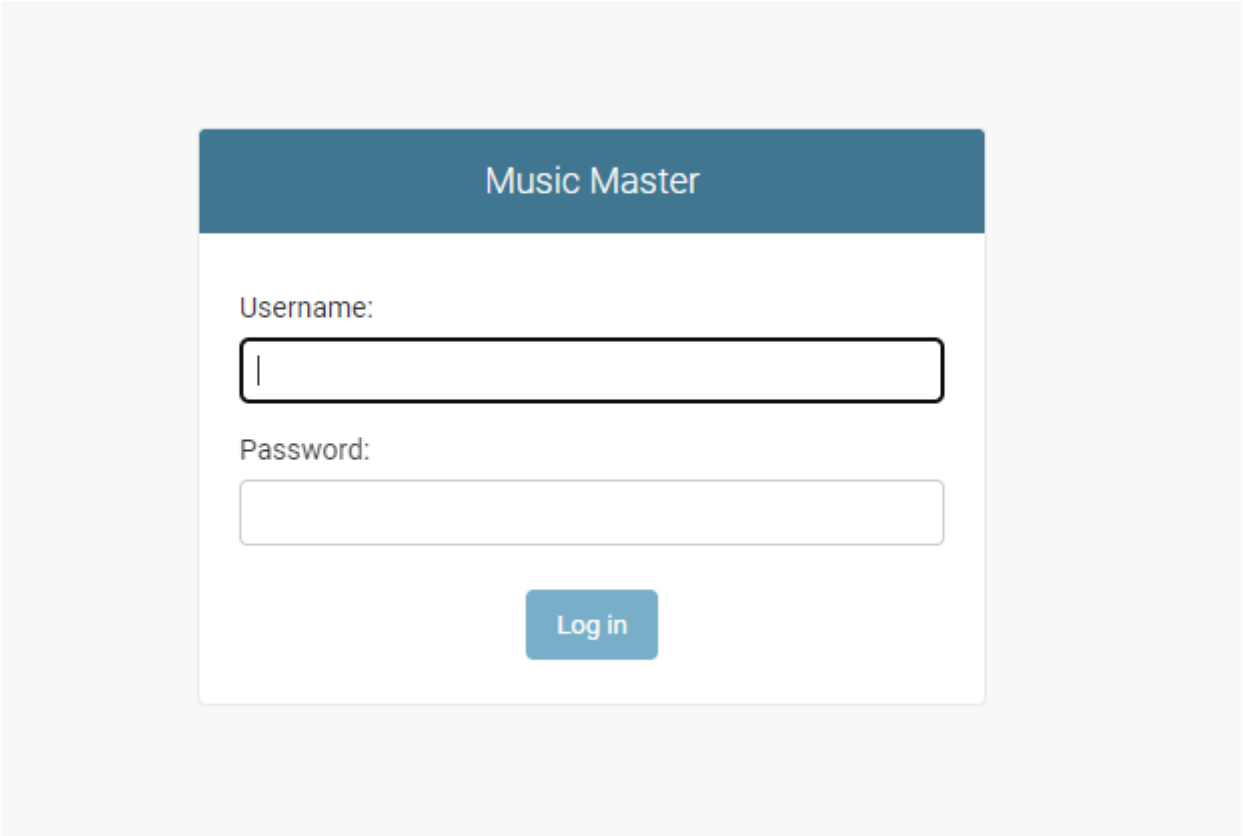

Рисунок 3.2 - Сторінка авторизації до системи «Music Master»

Після того, як ми правильно введемо наш логін та пароль, нас зустрічає головна сторінка сервісу, яка представлена на рисунку 3.3.

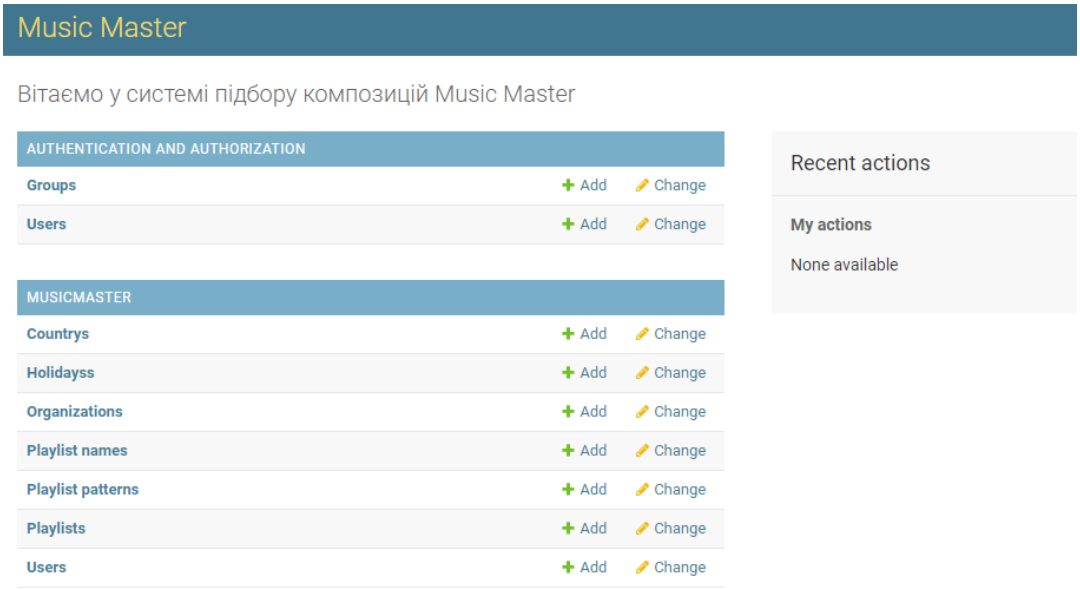

Рисунок 3.3 - Головна сторінка системи MusicMaster

Нам потрібно наповнити певні таблиці перед початком тестування нашого серця сервісу – алгоритму підбору музики. В нашому випадку, завдяки Django, наповнення відбувається зручним чином. Розглянемо за приклад наповнення таблиці Holidays. На рис.3.4 представлено меню таблиці Holidays.

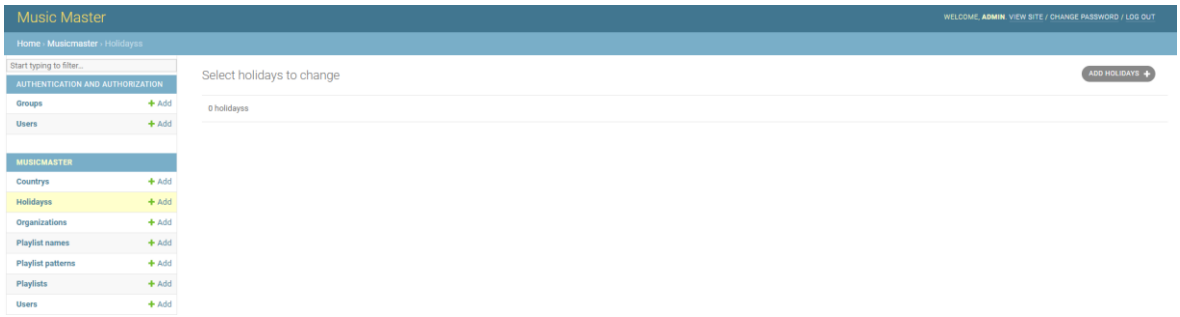

Рисунок 3.4 - Меню таблиці Holidays

Натискаємо на кнопку “Add Holidays” і ми потрапляємо до меню створення свята (рис.3.5).

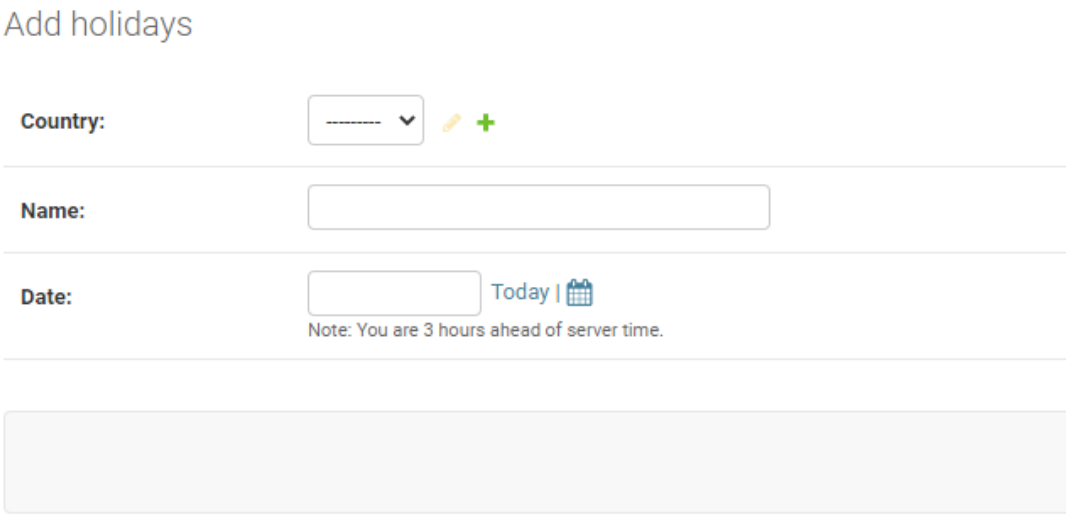

Рисунок 3.5 - Меню додавання свята

Бачимо, що нам потрібно обрати країну, ім’я та дату, коли буде свято. Проте, є нюанс: ми ще не наповнили таблицю з країнами, тому теоретично створення свята без наповненої таблиці країн неможливе. Фреймворк Django навіть це передбачив, тому прямо з одного меню є можливість створити дані в іншій таблиці, якщо це foreign key (рис.3.6).

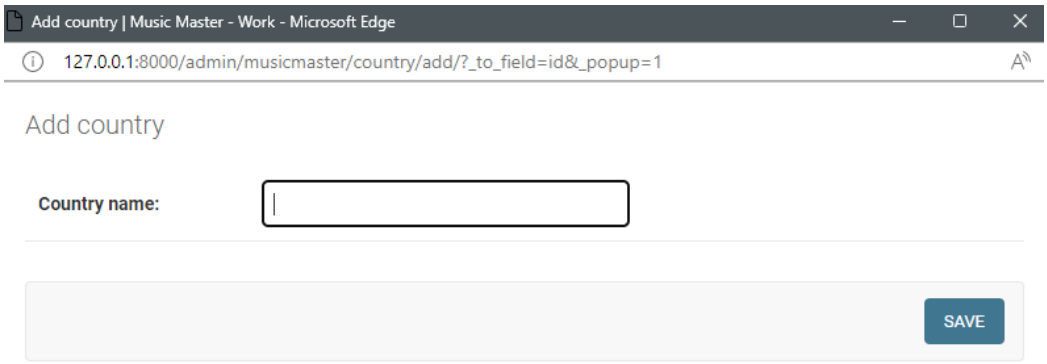

Рисунок 3.6 - Створення країни в таблиці Countries, не виходячи з меню створення свята

Ми заповнюємо меню потрібними даними (рис.3.7), натискаємо на кнопку SAVE і потім перевіряємо, чи коректно додалися дані в таблицю.

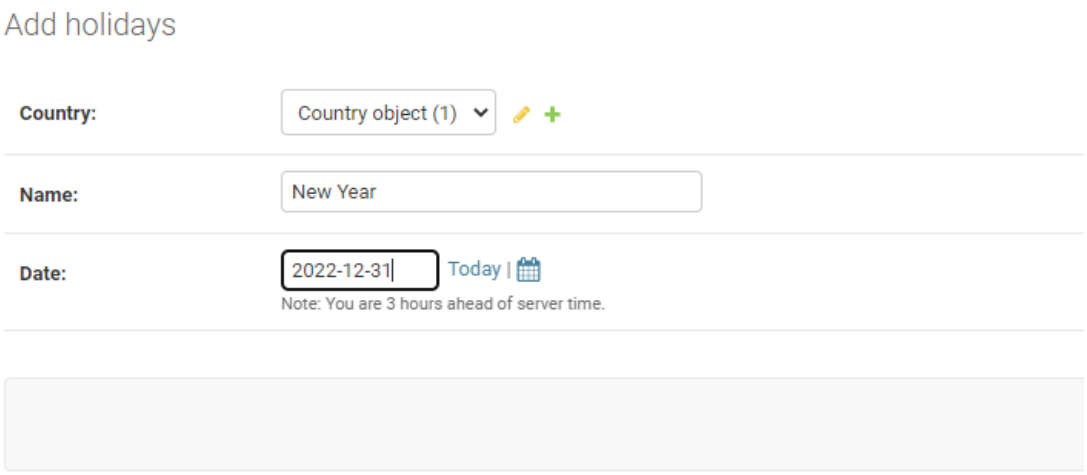

Рисунок 3.7 - Заповнення полів для додавання даних в таблиці Holidays

Перевіряємо таблицю Holidays і, як бачимо, дані додалися коректно (рис.3.8).

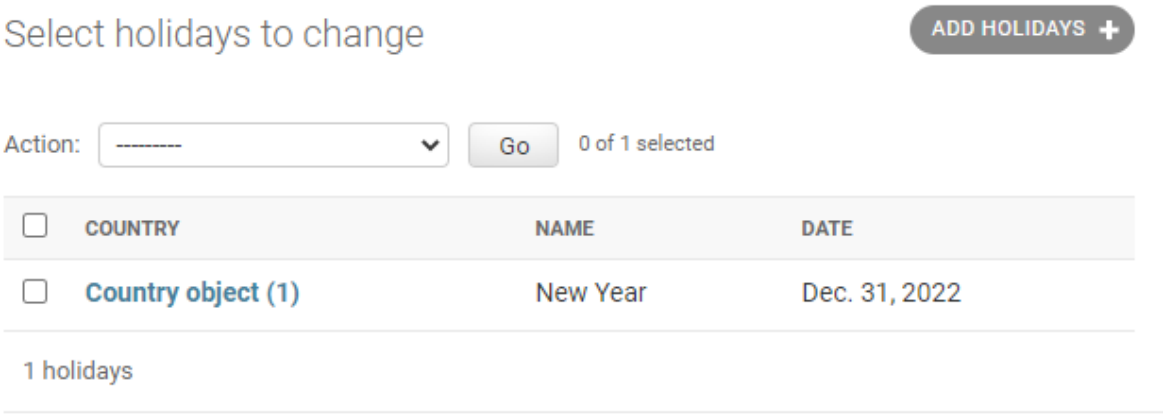

Рисунок 3.8 - Дані в таблиці Holidays після створення

Наповнюємо інші таблиці потрібними даними для коректного тестування застосунку і переходимо до тестування його головної функції – підбору музики.

Переглянути код серця цього веб-застосунку можливо буде у додатку А, а зараз демонструємо працездатність даного алгоритму. На рис.3.9 представлено заповнену таблицю треками, які були підібрані за допомогою нашого алгоритму на Spotify.

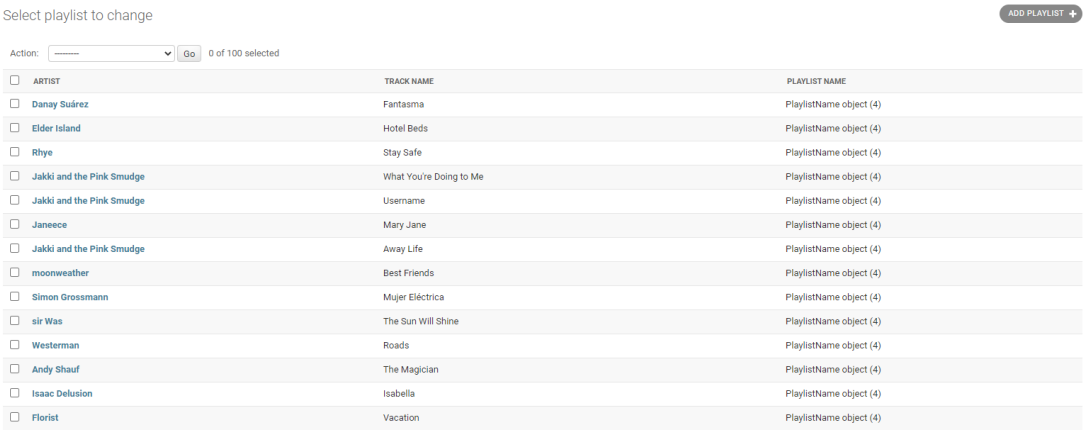

Рисунок 3.9 - Список треків, які були підібрані завдяки алгоритму

На рис. 3.10 та рис. 3.11 представлено таблицю назв плейлистів, а також таблицю шаблонів після того, як було виконано алгоритм пошуку треків згідно певних чинників.

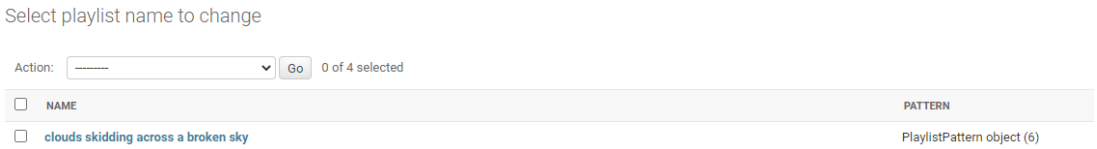

Рисунок 3.10 - Таблиця з іменем нового плейлиста та його шаблоном

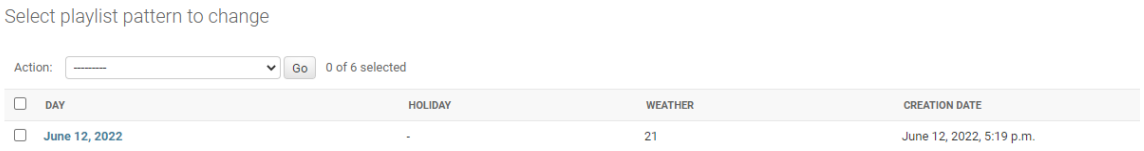

Рисунок 3.11 - Таблиця даними про новий шаблон

Отже, сервіс працює справно і ми можемо перейти до висновків.

ВИСНОВКИ

В ході виконання роботи були вирішені наступні задачі:

- Ознайомилися зі Spotify API
- Провели аналіз схожих рішень та тих рішень, на основі яких планувалося розробити власний
- Спроектовано алгоритм підбору музичних композицій, веб-застосунок та базу даних для нього
- проведено тестування працездатності алгоритму та веб-застосунку в цілому

Результатом виконання роботи є створений веб-застосунок MusicMaster, який допоможе власникам ТРЦ без зайвих зусиль підбирати якісну музику для їх об’єктів, а головне – за допомогою неї залучати більше покупців.

Розроблений застосунок має такі перспективи:

- перехід з початкового алгоритму до досконалого;
- перехід з SQLite на PostgreSQL;
- додавання інструментів для планування задач, таких як Celery, Redis;
- написання повноцінної версії застосунку з використанням React.js;

СПИСОК ЛІТЕРАТУРИ

1. Медіа-агенція Bolero [Електронний ресурс] – Режим доступу: http://www.bolero.kiev.ua/
2. Фонова музика для бізнесу Retail Music [Електронний ресурс] – Режим доступу : https://retail-music.com.ua/
3. Рішення для бізнесу Esthetic Sound [Електронний ресурс] – Режим доступу: https://esound.com.ua/
4. Spotify. Welcome to nirvana [Електронний ресурс] – Режим доступу : https://web.archive.org/web/20090121093939/http://www.guardian.co.uk/music/2009/jan/16/downloading-music-spotify
5. Youtube Music [Електронний ресурс] – Режим доступу : https://music.youtube.com/
6. Django 4.0.5. Documentation [Електронний ресурс] – Режим доступу : https://docs.djangoproject.com/en/4.0/
7. Чим PostgreSQL краще за інші СУБД з відкритим вихідним кодом. Частина 1 [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/282764/
8. Використання Redis [Електронний ресурс] – Режим доступу: https://django.fun/docs/celery/ru/5.1/getting-started/backends-and-brokers/redis/
9. REST Architectural constraints [Електронний ресурс] – Режим доступу: https://restfulapi.net/rest-architectural-constraints/
10. Django QuerySet API Reference [Електронний ресурс] – Режим доступу: https://docs.djangoproject.com/en/4.0/ref/models/querysets/
11. Django Tutorial. MDN Web Docs [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/skeleton_website
12. Bootstrap 5. Contents [Електронний ресурс] – Режим доступу: https://getbootstrap.com/docs/5.1/getting-started/contents/

13. Spotipy Documentation [Електронний ресурс] – Режим доступу: https://spotipy.readthedocs.io/en/2.19.0/
14. PyOWM Documentation [Електронний ресурс] – Режим доступу: https://pyowm.readthedocs.io/en/latest/
15. Templates. Django documentation [Електронний ресурс] – Режим доступу: https://docs.djangoproject.com/en/4.0/topics/templates/

ДОДАТОК

Фрагменти вихідного коду проекту

Playlist_engine.py

```
import spotipy
import pyowm
from .models import Playlist, PlaylistName,
PlaylistPattern

WTOKEN = ""
SPOTIPY_CLIENT_ID = ""
SPOTIPY_CLIENT_SECRET = ""

owm = pyowm.OWM(WTOKEN)
mgr = owm.weather_manager()


def playlist_engine(city):
    observation = mgr.weather_at_place(city)
    w = observation.weather
    temp = round(w.temperature('celsius').get('temp'))
    sp =
spotipy.Spotify(client_credentials_manager=spotipy.Spot
ifyClientCredentials(SPOTIPY_CLIENT_ID,

SPOTIPY_CLIENT_SECRET))
    results = sp.search(w.detailed_status,
type='playlist')
    items = results['playlists']['items']
    playlist_from_sp =
sp.playlist_items(items[0]['id'])
    playlist_pattern =
PlaylistPattern.objects.create(weather=temp)
    playlist_name =
PlaylistName.objects.create(name=items[0]['name'],
pattern=playlist_pattern)
    for i in range(0, 99):
        try:
            artist =
playlist_from_sp['items'][i]['track']['artists'][0]['na
me']
            track_name =
```

```
playlist_from_sp['items'][i]['track']['name']
            Playlist.objects.create(artist=artist,

track_name=track_name,

playlist_name=playlist_name)
        except IndexError:
            print("That's all tracks")
            break
```

admin.py

```
import pyowm
import spotipy
from django.contrib import admin

from .models import Holidays, Country, User,
Organization, PlaylistPattern, PlaylistName, Playlist

@admin.register(Country)
class CountryAdmin(admin.ModelAdmin):
    list_display = ("country_name",)


@admin.register(Holidays)
class CountryAdmin(admin.ModelAdmin):
    list_display = ("country",
                    "name",
                    "date")


@admin.register(Organization)
class OrganizationAdmin(admin.ModelAdmin):
    list_display = ("name",
                    "country",
                    "city",
                    "start_work",
                    "end_work",
                    "email",
                    "phone_number",
                    "ad_integration")
```

```
@admin.register(User)
class UserAdmin(admin.ModelAdmin):
    list_display = ("username",
                    "password",
                    "phone_number",
                    "organization")


@admin.register(PlaylistPattern)
class PlaylistPatternAdmin(admin.ModelAdmin):
    list_display = ("day",
                    "holiday",
                    "weather",
                    "creation_date")


@admin.register(PlaylistName)
class PlaylistNameAdmin(admin.ModelAdmin):
    list_display = ("name",
                    "pattern")


@admin.register(Playlist)
class PlaylistAdmin(admin.ModelAdmin):
    list_display = ("artist",
                    "track_name",
                    "playlist_name")
```

models.py

```
from django.db import models


class Country(models.Model):
    country_name = models.CharField(max_length=25)


class Holidays(models.Model):
    country = models.ForeignKey(Country,
on_delete=models.CASCADE)
    name = models.CharField(max_length=25)
    date = models.DateField()


class Organization(models.Model):
```

```
    name = models.CharField(max_length=25)
    country = models.ForeignKey(Country,
on_delete=models.CASCADE)
    city = models.CharField(max_length=25)
    start_work = models.TimeField()
    end_work = models.TimeField()
    email = models.EmailField()
    phone_number = models.IntegerField()
    ad_integration = models.BooleanField()


class User(models.Model):
    username = models.CharField(max_length=30)
    password = models.CharField(max_length=30)
    phone_number = models.IntegerField()
    organization = models.ForeignKey(Organization,
on_delete=models.CASCADE)


class PlaylistPattern(models.Model):
    day = models.DateField(auto_now_add=True,
blank=True)
    holiday = models.ForeignKey(Holidays,
on_delete=models.CASCADE, null=True)
    weather = models.CharField(max_length=30)
    creation_date =
models.DateTimeField(auto_now_add=True, blank=True)


class PlaylistName(models.Model):
    name = models.CharField(max_length=30)
    pattern = models.ForeignKey(PlaylistPattern,
on_delete=models.CASCADE)


class Playlist(models.Model):
    artist = models.CharField(max_length=30)
    track_name = models.CharField(max_length=30)
    playlist_name = models.ForeignKey(PlaylistName,
on_delete=models.CASCADE)
```

urls.py

```
"""musicmaster URL Configuration

The `urlpatterns` list routes URLs to views. For more
information please see:

https://docs.djangoproject.com/en/4.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home,
name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('',
Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls
import include, path
    2. Add a URL to urlpatterns:  path('blog/',
include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]

admin.site.site_header = "Music Master"

admin.site.site_title = "Music Master"

admin.site.index_title = "Вітаємо у системі підбору
композицій Music Master"
```